Multi-Dimensional Context-Aware Adaptation of Service Front-Ends

Project no. FP7 – ICT – 258030

Deliverable 3.3.1
AAL-DL: Semantics, Syntaxes and Stylistics (R1)

Due date of deliverable: 29/02/2012
Actual submission to EC date: Add EC date

Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013)

| Dissemination level | [PU] | [Public] | Yes |

This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA (This license is only applied when the deliverable is public).
### Document Information

<table>
<thead>
<tr>
<th></th>
<th>ISTI-CNR</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Lead Contractor</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Editor</strong></td>
<td>Carmen Santoro, Fabio Paternò, Davide Spano</td>
</tr>
<tr>
<td><strong>Revision</strong></td>
<td>Ignacio Marin</td>
</tr>
<tr>
<td><strong>Reviewer 1</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Reviewer 2</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Approved by</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Project Officer</strong></td>
<td></td>
</tr>
</tbody>
</table>

### Contributors

<table>
<thead>
<tr>
<th>Partner</th>
<th>Contributors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Changes

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01/02/2012</td>
<td>ISTI-CNR – Carmen Santoro</td>
<td>Structure of the document</td>
</tr>
<tr>
<td>2</td>
<td>23/02/2012</td>
<td>Carmen Santoro</td>
<td>First version</td>
</tr>
<tr>
<td>3</td>
<td>29/02/2012</td>
<td>Carmen Santoro</td>
<td>Consolidated version</td>
</tr>
</tbody>
</table>
Executive Summary

The main goal of this deliverable is to present the AAL-DL, a high-level description language intended to express declaratively advanced adaptation logic.
# Table of Contents

1 Introduction .......................................................................................................................... 5  
  1.1 Objectives ......................................................................................................................... 5  
  1.2 Audience ........................................................................................................................... 5  
  1.3 Related Documents and their Relationships with the Current Document ......................... 5  
  1.4 Organization of This Document ......................................................................................... 5  

2 The Advanced Adaptation Logic Description Language (AAL-DL) .............................................. 6  
  2.1 Motivations ........................................................................................................................ 6  
  2.2 Related Work ..................................................................................................................... 6  
  2.3 Description of the AAL-DL ............................................................................................... 7  
    2.3.1 Requirements of the AAL-DL ..................................................................................... 7  
    2.3.2 The MetaModel ........................................................................................................... 8  
  2.4 The Associated XML Schema .......................................................................................... 9  

3 Some Modelling Examples ...................................................................................................... 16  
  3.1 Referring external models within adaptation rules ............................................................... 16  
  3.2 Example 1 ........................................................................................................................ 17  
  3.3 Example 2 ........................................................................................................................ 19  
  3.4 Example 3 ........................................................................................................................ 21  
  3.5 Example 4 ........................................................................................................................ 22  
  3.6 Example 5 ........................................................................................................................ 23  

4 Conclusions ............................................................................................................................ 25  
  4.1 Final Remarks and Future Work ...................................................................................... 25  

References .................................................................................................................................. 26  
Acknowledgements ................................................................................................................... 27  
Glossary ..................................................................................................................................... 28
1 Introduction

1.1 Objectives
This deliverable describes the characteristics of the AAL-DL language, which will be used to specify the adaptations applied on the UI-DL for obtaining context-aware interactive applications.

1.2 Audience
This document has a public dissemination level, so theoretically it is open to public consultation by the general public. However, a key audience is represented by Project reviewers and officer, as well as any researcher/scientist who could be interested in the topics addressed by SERENOA.

1.3 Related Documents and their Relationships with the Current Document
- **D1.1.2 (Requirements Analysis (R2))**: Deliverable D1.1.2 provides the requirements for the various modules of the SERENOA architecture, including the AAL-DL. D1.1.2 also gives an overview of the current state of the implementation of requirements for the AAL-DL.
- **D1.2.1 (Architectural Specifications (R1))**: Deliverable D1.2.1 provides an overview of the architecture of the platform that is currently being considered in SERENOA: in D1.2.1 document you can find how the AAL-DL fits this architecture and its role.
- **D3.2.1 (ASFE-DL: Semantics, Syntaxes and Stylistics (R1))**: Deliverable D3.2.1 describes the language for specifying the interactive application (yet currently focusing only to the abstract level). References to this D3.2.1 deliverable (and also to further releases of the ASFE-DL) are necessary so that the two description languages (AAL-DL and ASFE-DL) grow in parallel and share as much semantic and syntactic features as possible.
- **D4.4.1 (Context of Use Runtime Infrastructure (R1))**: Deliverable D4.4.1 provides an overview of how the context is managed in SERENOA. More specifically, since modelling the context is essential for expressing adaptation rules, references to this deliverable D4.4.1 have to be done within the current document (e.g. for the specification of the context model).

1.4 Organization of This Document
This deliverable is organised in the following way: Chapter 2 describes the AAL-DL also providing some motivations and related work, Chapter 3 provides some relevant modelling examples, Chapter 4 summarises conclusions and provides insights for future work.
2 The Advanced Adaptation Logic Description Language (AAL-DL)

2.1 Motivations

One of the objectives of the project is to develop a high-level description language intended to express declaratively advanced adaptation logic (AAL). In this document we mainly describe the characteristics of this language, which will be used to specify the adaptations to be applied upon the logical descriptions of interactive applications for obtaining context-aware interactive applications.

This language can be an important integration point among the components developed in the project, since it provides a declarative indication of what contextual events should be considered and the consequent modifications that should be performed on the interactive applications that then should be rendered on various platforms.

2.2 Related Work

The AAL Description Language is related to other works/languages for model transformations.

For instance, QVT (Query/View/Transformation) is a standard set of languages for the transformation of models, defined by the Object Management Group\(^1\) (OMG). Its specification provides a hybrid approach for the transformation definition, providing both declarative and imperative constructs. The declarative level is in turn composed of two languages, at different level of abstraction but with the same expressive power. The top layer is the Relation metamodel, which support complex pattern matching and template creations, together with the creation of implicit traces between the starting and the target model. Such high-level constructs are translated into a more compact language, which defines a minimal extension of the Essential Meta Object Facility (OMG, 2011) and the Object Constraint Language (OMG, 2012), with object creation, deletion and traces explicitly defined. The imperative syntax of the Operation Mappings defined into the specification, allows the definition of transformation using constructs familiar to imperative programmers (assignments, if-then-else, while etc.). The specification allows also the definition of Black Box Transformations.

The ATL (Atlas Tranformation Language) (Jouault et al., 2006) is an implementation of the QVT proposal by OBEO and INRIA. It maintains the possibility to specify the transformation with a set of declarative matched rules that transform elements of the source model into elements of the target model. They are triggered with a pattern matching mechanism. In order to query the source model, it is possible to exploit the OCL (OMG, 2012) language. In addition, the language provides the possibility to invoke the called rules, which allow the definition of a part of the transformation through an imperative syntax.

XSLT (eXtensible Stylesheet Language Transformations) (W3C, 2007) is an XML syntax for defining transformations starting with XML files to different text formats, obviously including XML itself, based on pattern matching. A style sheet defines a set of rules that allows to take as input an XML tree (source) and to transform it into a destination tree (result). The analysis of the source document syntax is performed by the XSLT processor that applies the rules on elements that match a given template.

These three languages for transformations are not specific to describing context-dependent adaptation logics but can express model transformations and we consider model-based descriptions of interactive applications, thus can provide useful concepts and can be considered for implementing some of the adaptations required.

Another work which is interesting to mention is a recent publication appeared in Communications of the ACM journal (Lee et al., 2012). In this paper authors present a mobile context-monitoring framework intended to address the resource challenges of sensor-rich mobile environments in support of context-aware applications. MobiCon middleware lets mobile application developers leverage diverse user contexts without concern for the precision of context recognition or the battery and computational resources of smartphones and other sensor devices in context monitoring. MobiCon translates physical context into the most energy-

\(^1\)\url{http://www.omg.org/}
and-processing efficient combination of resources at runtime—the kind of technology needed by practical context-monitoring platforms or applications. As a middleware for context-monitoring APIs, MobiCon includes the Context Monitoring Query, or CMQ, declarative query language supporting rich semantics for monitoring a range of contexts while abstracting device details. It also proactively detects changes in user context. CMQ specifies three conditions: context, alarm, and duration. The context part contains information about the current context like e.g. location, activity, time, etc. Alarm determines when MobiCon delivers an alarm event to an application. Duration specifies the amount of time MobiCon must evaluate a registered CMQ. The CMQ translation process converts CMQs specified in context level semantics into range predicates over continuous feature data, along with associated sensors and corresponding resource demands. MobiCon maps context types specified in a registered CMQ to one or more features and their associated sensors; for example, it converts an activity context type into multiple features like direct current derived from accelerometers. A context value is then transformed into numerical value ranges for corresponding features; for example, “humidity == wet” can be mapped to “80%< humidity”. This work is interesting for us as it also tries to model context changes for supporting context-aware applications through the use of a declarative query language. In the next section we will more properly focus on the features of the description language we identified in Serenoa.

2.3 Description of the AAL-DL

The AAL-DL is a high-level description language intended to express declaratively advanced adaptation logic (AAL). Such adaptation logic should define the transformations affecting the interactive application when some specific situations occur both at the context level (e.g. an entity of the context changes its state), and in the interactive application (e.g. an UI event is triggered).

In the current specification of the AAL-DL we have considered the definition of first-order adaptation rules (simple adaptation rules like e.g., adapt this SFE for this platform) and second-order adaptation (rules that govern the application of adaptation rules e.g. by selecting first-order rules): indeed, in our language the action part of a rule can be in turn another rule. We have not yet considered third-order adaptation rules (strategies that privilege some adaptation approach for usability, performance, reliability i.e. rules that promote or demote sets of second-order rules).

2.3.1 Requirements of the AAL-DL

The main requirements that have driven the building of the AAL-DL have been already described in D1.2.1. Here we briefly recall them:

- **Compliance with ECA format:** The rules in the AAL-DL are expressed through an ECA-based (Event, Condition, Action) format.
- **Specification of rules:** The rules in the AAL-DL support specifications of (Event, Condition, Action) rules in which: i) events are changes of context state or UI state; ii) conditions are Boolean predicates referring to context state or UI state; iii) actions are changes in the interactive application.
- **Support for different UI abstraction levels:** The rules in the AAL-DL refer to different levels of UI abstractions.
- **Support for elementary and complex events:** The “event” part of a AAL-DL rule can be either an elementary event or a complex event (e.g. composed through some Boolean operators).
- **Support for elementary and complex conditions:** Conditions appearing in a AAL-DL rule can be either elementary (e.g. atomic Boolean predicates) or complex conditions (namely: obtained by exploiting Boolean operators).
- **Support for single action and sequence of actions:** In the “action” part of a rule there is either one single action or a sequence of actions.
- **Support for rule priority:** Rules of the AAL-DL have priorities, which are useful when multiple, conflicting rules occur simultaneously, thus priorities act as a mechanism to identify the rule which is the most likely to be triggered
- **Support for rule nesting:** In the AAL-DL, rules can be nested: the action of a meta-rule can be a set
of rules.

- **Support for rule inheritance:** The AAL-DL supports inheritance between the various rules: a general rule could have different sub-classes.
- **Notation:** The AAL-DL is formalised according to a well specified syntax.

The AAL-DL should be able to model transformations to be performed on the interactive application when context-aware adaptations are needed. Such specific situations can be due to changes at the context level (e.g. an entity of the context changes its state), and/or in the interactive application (e.g. an UI event is triggered). In the following section we describe the XML-based language we identified for expressing such logic.

It is worth pointing out that, in order to specify such transformations, the AAL-DL has also to refer to entities that might be defined in other models (e.g. the context model, the UI model, etc.). Therefore, appropriate mechanisms for referencing from the AAL-DL the models specifying other key aspects of the framework (e.g., the context, the UI) should be properly identified and supported. In the following sections we will explain how we supported this in our approach.

### 2.3.2 The MetaModel

In Figure 1 there is the UML class diagram representing the main classes, interrelationships, and attributes of the AAL-DL. The description of this metamodel will be done in more details in the next section, through the description of the specific XML-based language that we have identified so far for implementing its concepts (see Section 2.4).

As you can see from Figure 1, at the highest level we have the element `ruleModel`, which will contain a number of `rules`, each one consisting of a triplet `<event, condition, action>`. Events are those occurring not only in the interactive application but also e.g. in the context. Conditions are Boolean statements that have to be satisfied (in order to trigger the execution of actions). Actions are all the actions that can occur in an interactive application.

![Figure 1: The class diagram representing main concepts and relationships of AAL-DL](image)

It is worth pointing out that the complete specification of the `action` element has not been represented in Figure 1. In Figure 1 only the main subclasses of `action` element (create, read, update, delete, if, while,
foreach, for, block, invokeFunction) have been included. This has been done as the specification of the actions is shared between the AAL-DL and the ASFE-DL since the actions considered in the AAL-DL are exactly the actions that can occur in the interactive application and therefore modelled in the ASFE-DL. We have made this choice in order to avoid redundant definitions and, more importantly, inconsistent specifications between the two languages.

2.4 The Associated XML Schema

In this section we describe the XML Schema identified so far for the AAL-DL. At the root level of the AAL-DL we have an object named ruleModel, having type named RulesType (see Figure 2).

```xml
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">

   <xs:element name="ruleModel" type="RulesType"/>

</xs:schema>
```

Figure 2. The root node of the AAL-DL XML Schema: ruleModel

RulesType is a complex type which defines the structure of a document describing the adaptation rules. Its XML-based specification follows in Figure 3. Please note that, in this figure, an annotation element also appears: its content has been folded/hidden as it just contains explanatory comment of the associated specification).

```xml
<xs:complexType name="RulesType">
   <xs:annotation>
      <xs:sequence>
         <xs:element name="ext_model_ref" type="ExternalModelReferenceType" maxOccurs="unbounded"/>
         <xs:element name="rule" type="RuleType" maxOccurs="unbounded"/>
      </xs:sequence>
   </xs:complexType>
```

Figure 3. The specification of RulesType

RulesType consists of a sequence of two elements:

- `ext_model_ref`, is a reference to an external model that can be referred by the adaptation rule. It has type named ExternalModelReferenceType. The maximum occurrence of such kind of elements is unbounded since multiple external models can be referred by adaptation rules. Indeed, since with adaptation rules we want to model actions for reacting to context or UI events when certain conditions hold, it is clear that we need to refer to information contained in UI models (e.g. abstract or concrete) and also into the context model. All such information should be specified in separate documents. Therefore the `ext_model_ref` element should provide the information about where we can find such separate models referred by the rules.

- `rule` is the element modelling the proper adaptation rule, which has type named RuleType (its definition follows in the continuation of this section).

The ExternalModelReferenceType (see Figure 4) is a complex type used to refer to external resources (e.g. other models) from the current document specifying the adaptation rules. In particular, the ExternalModelReferenceType type consists of two attributes: `URI`, which defines the URI of the referred external resource (e.g. the path where the UI model can be found); and `model_id`, which specifies an identifier to be used in the current document for easily referring the concerned resource (e.g.: the referred, external UI model).
The RuleType is a complex type defining the structure of an adaptation rule. It consists of a sequence of three elements (see Figure 5):

- A part specifying an event, expressed according to EventType specification. This part of the rule should describe the event whose occurrence triggers the evaluation of the rule. As we will see later on, this part could specify elementary events occurring in the interactive application, or even a composition of events. In any case, only one element (either an elementary event or an expression combining multiple events) should be defined (see minOccurs and maxOccurs attributes, both set to ‘1’ value).
- A part specifying a condition, described according to ConditionType. The condition of a rule is represented by a Boolean condition that has to be satisfied in order to execute the associated rule action(s). Since the condition could even not be specified, its minimum occurrence is 0.
- A part specifying the action of the rule. Actually, in the action part there might be either (1 to an unbounded number of) simple actions occurring in the interactive application or even (1 to an unbounded number of) other adaptation rules. In the first case, the action will directly specify the transformation/adaptation to be executed (on event’s occurrence and if the condition is satisfied); in the second case (if a new rule is specified in the “action” part), the step(s)/action(s) to do will depend on the evaluation of the nested rule.

In addition, a rule could have an optional priority attribute (modelled through an integer value), to identify the rule which is the most likely to be triggered when multiple, conflicting rules occur simultaneously.

An event has type named EventType (see Figure 6). An event could be either a simple event (having type named SimpleEventType) or an articulated expression specifying the composition of multiple events through Boolean operators (and having type named ComplexEventType).
The SimpleEventType (see Figure 7) can optionally define an agent that is responsible for the activation of the event. For instance, if we are modelling the fact that the user has changed his/her position, in this case the agent of the event is the user; if the concerned event regards the fact that device screen changes its state after a certain amount of time and goes to stand-by mode, in this case the agent of this event is the device itself. In addition, the attribute called event_name provides the name of the event, while the attributeGroup named EntityReferenceAttributeGroup will provide the information regarding the entity affected by the event itself.

The definition of the type named EntityReferenceAttributeGroup is included in the definition of the type named EntityReferenceType (see Figure 8) and consists of two attributes: xPath, which defines the xPath definition pointing to the concerned referred model; and externalModelId, which specifies the id of an external model, if the referenced entity is external to the current model.

Since an event could be specified through a Boolean expression combining multiple events, we included the definition of a type named ComplexEventType (see Figure 9). In this case, the specification will include at least two (or even more) elements of type EventType, combined through a Boolean operator having type named BooleanOperatorType.
The condition part of a rule has type named *ConditionType*. The specification of the *ConditionType* is described in Figure 10. It is worth pointing out that the specification of this type of elements (namely: conditions) has been included in another, separate XSD Schema (*Actions.xsd*, the XSD Schema modelling UI actions, which can be provided on request). This has to be done since the specification of conditions are not only needed for specifying adaptation rules, but also for specifying actions in the interactive application, therefore it has been judged useful to share their definition.

As you can see from Figure 10, the specification of *ConditionType* consists of a choice between various element options, depending on which entity (or entities) the condition involves: an *entityReference*, a *constant*, a *condition*, and an *expression*. In addition, there is also an attribute (*operator*), which should refer to a Boolean operator (*BooleanOperatorType*), used to combine multiple conditions. More in detail:

- *entityReference* has type named *EntityReferenceType* (defined above, see Figure 8). It should be used when the condition refers to a specific entity already defined somewhere else (e.g. a UI element, a context element).
- *constant* defines a constant value;
- *condition* has type named *ConditionType*. It has to be used when we want to nest conditions.
- *expression*, having name *ExpressionType*. It has to be used when the entity referred by the condition is not directly available but it is the result of a calculation, by using some operations (e.g. +, -, *, /, ..) within expressions.

The specification of the type named *ExpressionType* is defined below, together with the specification of *ExpressionOperator* and *BooleanOperatorType*. 

```xml
<xsd:complexType name="ConditionType">
  <xsd:choice minOccurs="1" maxOccurs="unbounded">
    <xsd:element name="entityReference" type="EntityReferenceType"/>
    <xsd:element name="constant" type="ConstantType"/>
    <xsd:element name="condition" type="ConditionType"/>
    <xsd:element name="expression" type="ExpressionType"/>
  </xsd:choice>
  <xsd:attribute name="operator" type="BooleanOperatorType"/>
</xsd:complexType>
```

**Figure 9. The specification of the ComplexEventType**

**Figure 10. The specification of the ConditionType**

The specification of the *ComplexEventType* is described in Figure 9. It consists of an element *event* of type *EventType*, with the condition part having type *ConditionType*.
The action part of a rule has BlockType as its type (see Figure 12). As it has been already mentioned, the definition of the action part of an adaptation rule is actually included in a separate XSD schema (Actions.XSD). This has been done since we noticed that the actions that can be carried out within an adaptation rule are the same actions that can be performed in an interactive application. Therefore, in order not to have redundancy between models and to have consistent definitions of parts that are shared between models, we decided to have only one definition provided within a XSD Schema and then reuse such a definition when needed in multiple places (namely: schemas).

In addition, please note that, since at the current moment only the abstract UI level has been specified in the Serenoa ASFE-DL, we had to refer to its abstract UI constructs within the specification of our rules. However, since the adaptation rules are expected to be able to refer to various UI abstraction levels (see in Section 2.3.1 the requirement titled “Support for different UI abstraction levels”), when the other abstraction levels will be covered in Serenoa, we can easily refer to a refinement of such actions and then be able to specify such rules referring to other abstraction levels (e.g. the concrete one).

The BlockType can assume various types (see Figure 12), depending on the action that we want to model. We
have included basic actions (like create, read, update, delete, invokeFunction), as well as more articulated constructs (like if, while, foreach, for, block). As the meaning of these actions is easy to be understood, in this document we include the specification of just one type of action (e.g. the create, see Figure 13). The complete specification of actions can be provided on request.

The create action consists of a number of elements (see Figure 13): the reference to the model entity (e.g. the UI container) that will contain the entity that is going to be created (containingEntityReference); the type of the element that will be created (TypeReferenceGroup); and the optional initial value (value) for initialising the newly created element. In addition, the create action has an attribute (entity_id) containing the reference to the new element, just created.

Lastly, we have also modelled inheritance: a general rule could have different sub-classes depending on e.g. the level of application (e.g.: level M1, M2 or M3).

Indeed, it could be useful in some cases to be able to take the definition of an existing rule, and then extend it to add more specific information. This is a classical mechanisms that in various development languages is called “inheritance” or “sub classing” (alternatively, there is the restriction mechanism). An example of extension for adaptation rules could be that in some cases we can define a rule that executes additional actions in comparison with another rule (which extends). In Figure 14 you can find how we specified this rule inheritance in our AAL-DL. However, this is just a draft idea and we plan to better detail this specification in a future release of this deliverable (D3.3.2), also providing some more concrete examples.
Figure 14. Specifying inheritance in the AAL-DL

```xml
<xs:complexType name="SpecialisedRuleTypeM1">
  <xs:annotation>
    <xs:complexType>
      <xs:extension base="RuleType">
        <xs:attribute name="M1"/>
      </xs:extension>
    </xs:complexType>
  </xs:annotation>
</xs:complexType>

<xs:complexType name="SpecialisedRuleTypeM2">
  <xs:annotation>
    <xs:complexType>
      <xs:extension base="RuleType">
        <xs:attribute name="M2"/>
      </xs:extension>
    </xs:complexType>
  </xs:annotation>
</xs:complexType>

<xs:complexType name="SpecialisedRuleTypeM3">
  <xs:annotation>
    <xs:complexType>
      <xs:extension base="RuleType">
        <xs:attribute name="M3"/>
      </xs:extension>
    </xs:complexType>
  </xs:annotation>
</xs:complexType>
```
3 Some Modelling Examples

In this section we describe some examples of modelling adaptation rules by exploiting the AAL-DL. Please note that some of such examples are related to M18 Demo. However, before going in detail in the specification of each example, in the next section we provide an explanation of how the references to external models have been concretely supported. As this mechanism is common to all the examples we briefly explain it and then we take it as implicit in all the examples. In general, any model can be referenced from the AAL-DL rules. In the examples that we provide below we consider the abstract model only because currently the SERENOAS ASFE-DL supports mainly this level. Thus, the examples describing modifications at the abstract level are only to exemplify the approach and other transformations between levels will be considered in the developments in the project as well.

3.1 Referring external models within adaptation rules

In Figure 15 you can find a draft overview of a document specifying adaptation rules complaint with our Serenno AAL-DL. As you can see, the document contains a couple of references to external models (the context model and the UI model, see the two elements named “ext_model_ref”), as well a partial specification of an adaptation rule (partial because for the moment we want just to focus on the mechanism for referring external resources). As you can see, the concerned adaptation rule refers to an entity in the context model (see the entityReference element in Figure 15). Since the context model is expected to be contained in a separate document, we need a cross-document mechanism to refer it.

More in detail, with reference to Figure 15:

- in (1) there is the location of the schema of the AAL-DL, which is referred by the current document that contains definitions of adaptation rules;
- in (2) it is defined the URI of the context model, which is referred by the current document.
- in (3) it is defined an ID for the context model. This ID will be used for convenience, in the current document, to refer the related model (i.e.: the context model).
- in (4) there is the reference to the external model through its defined ID.
- in (5) there is the xPath specification of the referred entity within the concerned (context) model.

Therefore, in the example shown in Figure 15, we need to refer in our condition to the category of a device (see (5) in Figure 15). The specification of this device is contained in the external model identified as “ctxModel” (see (4) in Figure 15). This “ctxModel” value is a reference to the (context) model which is
currently located at the location specified in (2), in Figure 15, and identified as “ctxModel” in (3), in the same Figure. In the following sections we will specify a number of example rules to show how, through this language, it is possible to specify some adaptation rules.

3.2 Example 1

Rule 1: If the device is a tablet, then master-detail groups will be rendered in one single view, else master is presented in one view and a new view is rendered when a detail is selected.

The rule considered here expresses that if the concerned mobile device is not a tablet, two separate presentations will be needed for two different UI groups: one presentation containing the UI part associated with the “master”, and another presentation containing the UI part showing the associated detail. Otherwise, if the device is a tablet, only one presentation will be needed to render both UI groups in the same presentation. In the rule, we describe the effects at the abstract level, obviously they could have been described at the concrete level as well.

```
<rule>
  <event>
    <simple_event event_name="onRender">
      <xPath>/AbstractUIModel/AbstractInteractionUnit[@id='AbstractUIModel/@current_unit']
        externalModelId="uiModel"/>
    </event>
    <condition operator="eq">
      <entityReference xPath="/context/technology/device/@category"
        externalModelId="ctxModel"/>
      <constant value="tablet" type="string"/>
    </condition>
    <action>
      <update>
        <entityReference xPath="/AbstractUIModel/@current_unit"
          externalModelId="uiModel"/>
        <value>
          <entityReference xPath="/AbstractUIModel/AbstractInteractionUnit[@id='singleView_presentation']/@id"
            externalModelId="uiModel"/>
        </value>
      </update>
    </action>
  </event>
</rule>

<rule>
  <event>
    <simple_event event_name="onRender">
      <xPath>/AbstractUIModel/AbstractInteractionUnit[@id='AbstractUIModel/@current_unit']
        externalModelId="uiModel"/>
    </event>
    <condition operator="neq">
      <entityReference xPath="/context/technology/device/@category"
        externalModelId="ctxModel"/>
      <constant value="tablet" type="string"/>
    </condition>
    <action>
      <update>
        <entityReference xPath="/AbstractUIModel/@current_unit"
          externalModelId="uiModel"/>
        <value>
          <entityReference xPath="/AbstractUIModel/AbstractInteractionUnit[@id='separateView_presentation']/@id"
            externalModelId="uiModel"/>
        </value>
      </update>
    </action>
  </event>
</rule>
```

Figure 16. The AAL-DL rules for modelling Rule 1
As you can see from the above AAL-DL specification, Rule1 is actually modeled by using two AAL-DL separate rules, shown in Figure 16.

The first AAL-DL rule specifies that, whenever the current presentation unit is loaded/rendered (see the event part of the first rule shown in that figure), if the category of the device is a tablet (see the condition part of the first rule shown in the figure), then the current_unit attribute (which should contain the ID of the currently active presentation) should be updated with the ID of the presentation (aka “AbstractInteractionUnit” in SERENOA ASFE-DL) identified as ‘singleView_presentation’. In this ‘singleView_presentation’ both the master and detail UI parts are included.

The specification of such AbstractInteractionUnit referred by the rule is expected to be contained in a separate model (the UI model).

The same is also true for the other case (see second rule in Figure 16), which models the case when the category of the device is not a tablet. In this case, the event part is the same as before. What changes is the condition: the current_unit attribute, which stores the ID of the currently active presentation, should be updated with the ID of the presentation (or AbstractInteractionUnit) having ‘separateView_presentation’ as its ID. This AbstractInteractionUnit will render only the master part in it. In a second presentation the detail part will be rendered. In order to reach this second presentation we expect that a navigator element will be contained in the separateView_presentation. Triggering this navigator will activate the rendering of the second presentation showing the concerned detail. A rough overview of how the presentations referred by Rule1 should be modelled is shown in Figure 17. Please note that in these specification we used the constructs of the Serenoa ASFE-DL.

In addition, in Figure 18 we show an example of context model instance for this example. The current context model visualized in this figure expresses the fact that the device named “my_device” is a tablet (therefore with this context state, the first rule visualized in Figure 16 should be triggered for adaptation.)
3.3 Example 2

**Rule 2:** *If user is colour-blind then use alternative color palette.*

The meaning of the rule is that, if the user is colour-blind, a different palette of colour should be exploited in the UI. The AAL-DL specification of this Rule 2 consists of two rules, shown in the following Figure 19. The first rule models the case when the user is colour-blind, whereas the second rule models the case when the user is not colour-blind.
In the first case (the user is colour-blind) the evaluation of the rule should be triggered whenever the AbstractInteractionUnit having the ID equal to the current AbstractInteractionUnit is loaded. This means that the evaluation of this rule is triggered on rendering the current presentation. The condition checks whether or not the user is colour-blind (this information should be properly contained in the context model therefore in the rule there is a reference to the concerned part in that model). The action describes the fact that on the above event, and if the condition is verified, then an update action should be executed, changing the currentPalette attribute of the interactive application with the current value of the attribute named alternativePalette. We expect that this alternative_palette object is defined within another model (the UI model).

In the second rule exploited for specifying Rule 2 we model the ‘default’ case. On rendering the current presentation and if the user does not have any type of blindness, then we have to restore the defaultPalette (namely: the currentPalette will be updated by actually referring to a defaultPalette, which, as before, we expect that it is modelled in a separate UI model).
3.4 Example 3

Rule 3: If battery_level < THRESHOLD then render still picture else render video

In this case we want to model that, if the battery level of the device goes under a certain threshold, then the adaptation should replace videos with alternative images in order to consume less battery power.

The AAL-DL specification of Rule 3 consists of two separate AAL-DL rules, shown in the following Figure 20 and Figure 21. Let’s consider the first AAL-DL rule.

When the battery level of the device changes (event), and if such a battery level is less than a certain predefined THRESHOLD (THRESHOLD is intended to contain a positive number) then we have to apply the following transformations to the UI: if in the interactive application some interactors of type video already exist then create, in the presentation currently containing the interactor of type video, another interactor for the image; then we update the uri of such an image with uri of the interactor named ‘stillPicture’ (which is the picture providing the alternative content for the video). Finally, the interactors having ‘video’ as their type should be removed (see the delete action in the AAL-DL specification).
3.5 Example 4

Rule 4: If user is dyslexic then use alternative font

The AAL-DL specification of this Rule 4 consists of two separate rules (both shown in Figure 22). The first one models the case when the user is dyslexic, the second rule models the case when the user is not dyslexic. In the first case the adaptation should simply update the value of the attribute named current_font with the value contained in the attribute named alternative_font (which should contain the font to be shown in case of user’s dyslexia). In the second case, the update should restore the default value for the font: therefore, the current_font should get the content of the attribute named default_font (which should contain the default value for the font).
3.6 Example 5

**Rule 5:** If user is near the post office and the post office is in its opening hours, and “pay a bill” task belongs to the user's TODO list, then show an alert/reminder presentation to the user

The AAL-DL specification of this Rule 5 is shown in Figure 23. The concerned event is the change of position by the user (detected through a change in his/her latitude or longitude). The condition that has to be satisfied consists of three sub-conditions that should be all verified (see the use of AND Boolean operator): the difference between the current user’s position and the position of the post office is less than a specific threshold, which means that the user is close to the post office); the post office has to be in its opening hours; and the task “pay bill” has to belong to the TODO list of the user.
If all such conditions are verified, then the action specified is to make the presentation showing an alert to the user as the current presentation (update action).

Figure 23: The AAL-DL specification of Rule 5
4 Conclusions

In this deliverable we have described the first version of the AAL Description Language for expressing adaptation rules. Initial concepts for specifying such rules have been included in the language and a XML-based Schema has also been provided for reference.

4.1 Final Remarks and Future Work

By Month 36 we plan to deliver an updated, second release of this deliverable (D3.3.2). In this next release we plan that the AAL-DL will better use and refer the concepts developed in the various project components (context infrastructure, ASFE-DL, CARFO ontology, …). For example, we will investigate how to connect the AAL-DL rules with the CARFO using OWL in order to perform queries to the knowledge base built from the ontology. In addition, in the next release of this document we plan to provide a broader and more refined set of examples will be provided in the future, and better address rules of second-order and third-order logic. Some initial draft ideas regarding second-order rules have already been identified in this version. In D3.3.2, further improvements of them are planned, as well as sketch out the design for third-order adaptation rules, so as to express adaptation in a more structured and powerful way.
References


Acknowledgements

- TELEFÓNICA INVESTIGACIÓN Y DESARROLLO, http://www.tid.es
- UNIVERSITE CATHOLIQUE DE LOUVAIN, http://www.uclouvain.be
- ISTI, http://giove.isti.cnr.it
- GEIE ERCIM, http://www.ercim.eu
- W4, http://w4global.com
- FUNDACION CTIC http://www.fundacionctic.org
Glossary

- http://serenoa.morfeo-project.org/glossary-of-terms